Rapport sur le traitement d'images de l'embolie pulmonaire

Réalisé par:
Oumaima EL MIAYAR

Encadré par:
Wafa Serrar

Liste des tableaux

Liste des figures

L'embolie pulmonaire

1- Définition de l'embolie pulmonaire

L’embolie pulmonaire se définit par l’obstruction d’une ou plusieurs artères pulmonaires plus ou moins distales par un embol qui peut être de composition diverse, entrainant une défaillance de la fonction pulmonaire.[1]

Ce blocage est le plus souvent causé par un caillot sanguin indiqué en fig 1 (phlébite ou thrombose veineuse) qui voyage jusqu’aux poumons à partir d’une autre partie du corps, très souvent à partir des jambes .

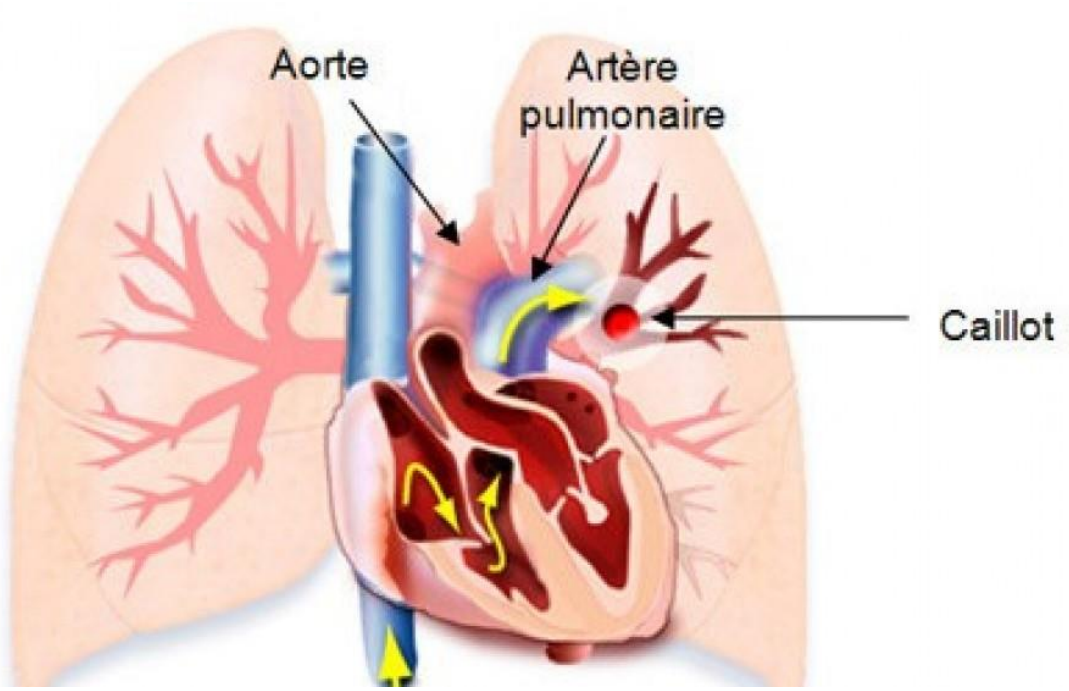


figure 1 :L'embolie pulmonaire

2-Causes de l'embolie pulmonaire

Un caillot sanguin qui se forme dans une veine profonde d’une jambe, du bassin ou d’un bras est appelé thrombose veineuse profonde. Lorsque ce caillot ou une partie de ce caillot voyage par la circulation sanguine jusqu’aux poumons, il peut entraîner un blocage de la circulation pulmonaire, c'est ce qu’on appelle une embolie pulmonaire.

Occasionnellement, une embolie pulmonaire peut être causée par des matières grasses provenant de la moelle osseuse d’un os cassé, des bulles d'air ou des cellules d’une tumeur.

3- Les symptômes de l'embolie pulmonaire

- Une douleur thoracique intense, qui peut ressembler aux symptômes d’une crise cardiaque et qui persiste malgré le repos.
- Un essoufflement soudain, des difficultés à respirer ou une respiration sifflante, pouvant survenir au repos ou à l’effort.
- De la toux, parfois accompagnée de crachats teintés de sang.

- Une sudation excessive (diaphorèse).
- De l'enflure généralement dans une seule jambe.
- Un pouls faible, irrégulier ou très rapide (tachycardie).
- Une coloration bleue autour de la bouche.
- Des étourdissements ou une syncope (perte de conscience).

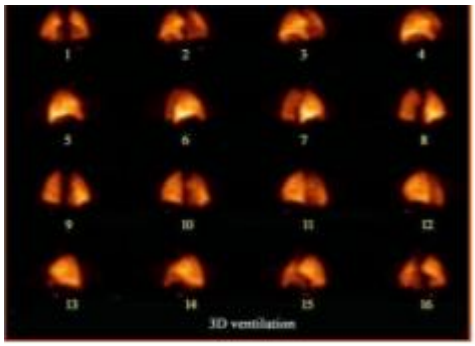
4- La diagnostique

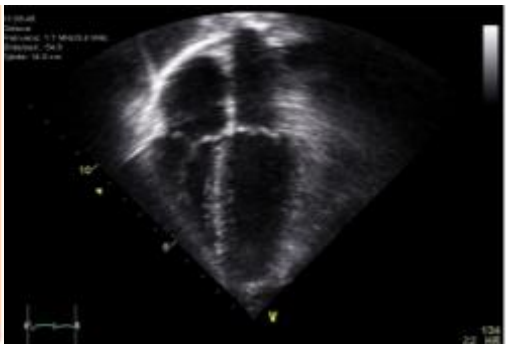
Chez les personnes qui présentent des maladies pulmonaires ou des maladies cardiovasculaires, il peut être difficile d'identifier la présence d'une embolie pulmonaire. Une série de tests, incluant des tests sanguins, une radiographie du thorax, une scintigraphie pulmonaire ou une tomodensitométrie des poumons peuvent aider à identifier la cause des symptômes.

Des radiographies du thorax sont presque toujours réalisées lorsque les médecins suspectent une maladie pulmonaire ou cardiaque. D'autres examens d'imagerie sont réalisés au besoin pour apporter aux médecins les informations spécifiques afin de poser un diagnostic.[2]

Les examens d'imagerie thoracique comprennent les suivants :

- Radiographies
- Tomodensitométrie (TDM)
- Imagerie par résonance magnétique (IRM)
- Tomoscintigraphie
- Echocardiographie
- Tomographie par émission de positrons (TEP)





Echocardiographie **Scintigraphie pulmonaire**

Pour notre projet ,nous allons utiliser des images Tomodensitomérie / CT scan (computed tomography scan) :

Pourquoi la Tomodensitomérie ?

La Tomodensitomérie (TDM) du thorax fournit des images plus précises qu'une simple radiographie.

Avec la TDM, une série de radiographies est analysée par un ordinateur, qui fournit alors plusieurs vues sous différents plans, telles que des vues longitudinales et des coupes transversales.

Tomodensitomérie

Au cours de la TDM, une substance visible sur les radiographies (agent de contraste radio-opaque) peut être injectée dans la circulation ou administrée par voie orale, afin de préciser la nature de certaines anomalies thoraciques. La TDM haute résolution et la TDM spiralée sont des procédures de TDM plus spécialisées.

La TDM haute résolution peut révéler plus de détails sur les maladies des poumons. La TDM spiralée peut fournir des images tridimensionnelles. En général, les TDM sont réalisées après la prise d'une inspiration profonde. Parfois, des images TDM sont obtenues après l'inspiration et l'expiration des personnes afin de mieux visualiser les petites voies respiratoires.[3]

L'embolie pulmonaire au coeur du traitement d'images

L’objectif général de l’imagerie médicale est de donner des informations sur les processus physiologiques du corps ainsi la détection de certains maladies qui peut affecter les organes , et l'embolie pulmonaire parmi ces maladies .

1- Le jeu de données : image DICOM

Avant de décrire les données utilisées , nous allons définir une image DICOM et sa relation avec le Standard DICOM

1-1- Standard DICOM

Le standard DICOM (Digital Imaging and COmmunications in Medicine), est un format de communication et d'archivage utilisé en imagerie radiologique. Le standard DICOM a été créé pour répondre à plusieurs objectifs :

- promouvoir l'interopérabilité entre différents appareils, par exemple un appareil IRM et une unité de visualisation externe, indépendamment des fabricants,
- faciliter le développement de systèmes d'archivage et de communication d'images(en anglais PACS, pour Picture Archiving and Communication System), s'interfaçant avec d'autres systèmes d'information hospitaliers,
- permettre la création de bases de données de diagnostics pouvant être interrogées à distance,
- mettre en œuvre le « dossier patient » incluant imagerie, comptes rendus, rapportsd'interventions, résultats d'analyses, hospitalisations, etc.

Pour ce faire, la norme DICOM définit un ensemble de protocoles et services réseau, ainsi qu'un format de stockage de données (image DICOM) et une structure de dossier médical (dossier patient)[4].

1-2 - Le jeu de données

Nous avons utiliser les données du site kaggle
https://www.kaggle.com/andrewmvd/pulmonary-embolism-in-ct-images/version/undefined

Les données sont tirées de l'article suivante :

Masoudi, M. et al. A new dataset of computed-tomography angiography images for computer-aided detection of pulmonary embolism. Sci. Data 5:180180 doi: 10.1038/sdata.2018.178 (2018).

L'ensemble de données actuel, FUMPE (qui signifie "ensemble de données sur l'embolie pulmonaire de l'université Ferdowsi de Mashhad") consiste en des images d'angiographie par Tomodensitométrie (CTA) pour l'embolie pulmonaire (EP) de 35 patients différents : Un total de 8 792 images CTPA d'une taille de 512×512 pixels . Les annotations ont été faites par

deux experts radiologues, prouvant la vérité de base à l'aide d'un outil logiciel de traitement d'images semi-automatisé.

2- Etapes du traitement d'images

2-1 les packages nécessaires

la première étape est de télécharger les packages nécessaires :

Nous avons utiliser la version python 3.7

le module **numpy** destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux;

le module **pydicom** pour la lecture du fichiers DICOM ;

le module **os** pour gérer l'arborescence des fichiers;

le module **matplotlib** pour la visualisation;

le module **skimage** est organisé en plusieurs sous-modules correspondant à plusieurs branches du traitement d'images : segmentation, filtrage, gestion des formats d'image;

le module **scipy** qui forme un regroupement de bibliothèques Python autour du calcul scientifique;

le module **plotly** pour les visualisations interactives

le module **ipywidget**s pour les reprénstations graphiques interactives.

2-2 Préparation des images DICOM

Pour cette étape , nous avons utiliser les données du patient d0003 , et nus allons choisit 236 DICOM images.

Nous commencons par le téléchargement des données ;

Nous vérifions , d'abord que nous sommes au bon répertoire

```
data_path = "fichier1"
output_path = working_path = "fichier2"
g = glob(data_path + '/*.dcm')

# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d DICOM images.\nFirst 10 filenames:" % len(g))
print ('\n'.join(g[:10]))
```

Nous avons obtenu le résultat suivant :

```
Total of 236 DICOM images.
First 10 filenames:
fichier1\D0200.dcm
fichier1\D0201.dcm
fichier1\D0202.dcm
fichier1\D0203.dcm
fichier1\D0204.dcm
fichier1\D0205.dcm
fichier1\D0206.dcm
fichier1\D0207.dcm
fichier1\D0208.dcm
fichier1\D0209.dcm
```

Ensuite, nous avons télécharger les images DICOM , en utilisant les deux fonctions suivants :

load_scan va charger toutes les images DICOM d'un dossier dans une liste pour les manipuler.

```
def load_scan(path):
    slices = [pydicom.read_file(path + '/' + s) for s in os.listdir(path)]
    slices.sort(key = lambda x: int(x.InstanceNumber))
    try:
        slice_thickness = np.abs(slices[0].ImagePositionPatient[2] - slices[1].ImagePositionPatient[2])
    except:
        slice_thickness = np.abs(slices[0].SliceLocation - slices[1].SliceLocation)

    for s in slices:
        s.SliceThickness = slice_thickness

    return slices
```

get_pixels_hu convertit les valeurs brutes en unités Houndsfeld .

```
def get_pixels_hu(scans):
    image = np.stack([s.pixel_array for s in scans])
    # Convert to int16 (from sometimes int16), should be possible as values should always be low enough (<32k)
    image = image.astype(np.int16)
    # Set outside-of-scan pixels to 1, The intercept is usually -1024, so air is approximately 0
    image[image == -2000] = 0
    # Convert to Hounsfield units (HU)
    intercept = scans[0].RescaleIntercept
    slope = scans[0].RescaleSlope

    if slope != 1:
        image = slope * image.astype(np.float64)
        image = image.astype(np.int16)

    image += np.int16(intercept)

    return np.array(image, dtype=np.int16)
id=0
patient = load_scan(data_path)
imgs = get_pixels_hu(patient)
```

Après , nous avons enregistrer le nouvel ensemble de données sur disque afin de ne pas avoir à retraiter la pile à chaque fois.

```
np.save(output_path + "fullimages_%d.npy" % (id), imgs)
```

2-3 Affichage des images

La première chose à faire est de vérifier si les unités de Houndsfeld sont à la bonne échelle et bien représentées.

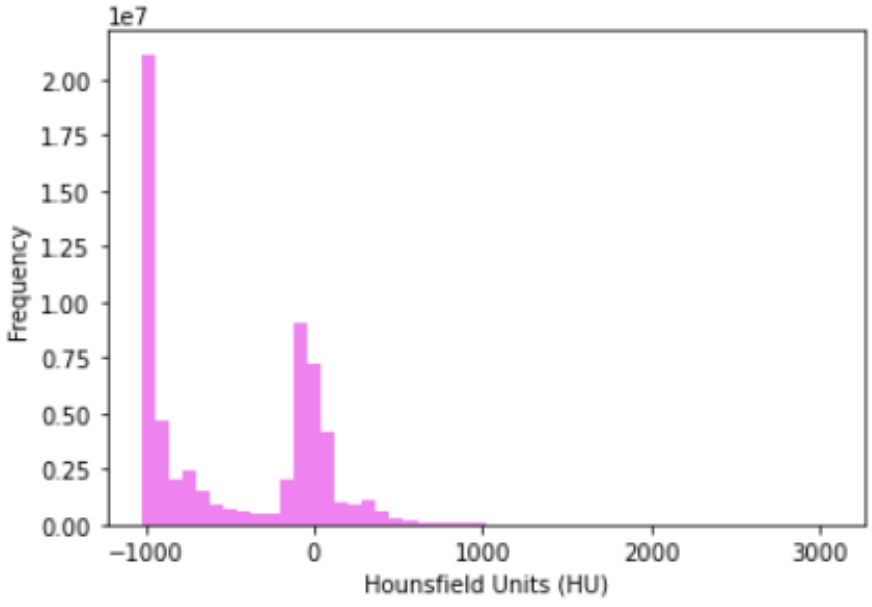
Les HU sont utiles car elles sont normalisées pour tous les scanners, quel que soit le nombre absolu de photons capturés par le détecteur du scanner. Si vous avez besoin d'une mise à jour, voici une liste rapide de quelques unités utiles, tirées de Wikipedia.

Substance	HU
Air	−1000
Lung	−500
Fat	−100 to −50
Water	0
Blood	+30 to +70
Muscle	+10 to +40
Liver	+40 to +60
Bone	+700 (cancellous bone) to +3000 (cortical bone)

Tableau 1: Exemples de densités d'absorption selon l'échelle de Hounsfield.

Créons maintenant un histogramme de toutes les données voxels

```
file_used=output_path+"fullimages_%d.npy" % id
imgs_to_process = np.load(file_used).astype(np.float64)

plt.hist(imgs_to_process.flatten(), bins=50, color='Violet')
plt.xlabel("Hounsfield Units (HU)")
plt.ylabel("Frequency")
plt.show()
```



Critique de l'histogramme

L'histogramme suggère ce qui suit :

- Il y a beaucoup d'air
- Il y a un peu de poumon
- Il y a une abondance de tissus mous, surtout des muscles, du foie, etc. mais il y a aussi un peu de graisse.
- Il n'y a qu'un petit morceau d'os (vu comme un minuscule éclat de hauteur entre 700 et 1000)

Cette observation signifie que nous devrons effectuer un prétraitement important si nous voulons traiter les lésions du tissu pulmonaire, car seule une infime partie des voxels représente le poumon.

Afficher une pile d'images

Nous n'avons pas beaucoup d'espace à l'écran, donc nous sauterons toutes les 2 tranches pour avoir un aperçu représentatif de l'étude.

```
id = 0
imgs_to_process = np.load(output_path+'fullimages_{}.npy'.format(id))

def sample_stack(stack, rows=7, cols=7, start_with=8, show_every=2):
    fig,ax = plt.subplots(rows,cols,figsize=[12,12])
    for i in range(rows*cols):
        ind = start_with + i*show_every
        ax[int(i/rows),int(i % rows)].set_title('slice %d' % ind)
        ax[int(i/rows),int(i % rows)].imshow(stack[ind],cmap='gray')
        ax[int(i/rows),int(i % rows)].axis('off')
    plt.show()

sample_stack(imgs_to_process)
```

Nons obtenons les images suivants.

L'air apparaît en gris parce qu'il a une valeur beaucoup plus élevée. Par conséquent, les poumons et les tissus mous ont également une résolution de contraste quelque peu réduite.

Nous essaierons de gérer ce problème lorsque nous normaliserons les données et créerons des masques de segmentation.

2-4- Echantillonnage

Bien que nous ayons chaque tranche individuelle, l'épaisseur de chaque tranche n'est pas immédiatement évidente.

Heureusement, cela se trouve dans DICOM header.

```
print ("Slice Thickness: %f" % patient[0].SliceThickness)
print ("Pixel Spacing (row, col): (%f, %f) " % (patient[0].PixelSpacing[0], patient[0].PixelSpacing[1]))
```

```
Slice Thickness: 0.500000
Pixel Spacing (row, col): (0.585938, 0.585938)
```

Cela signifie que nous avons des tranches de 0,5 mm, et chaque voxel représente 0,585938 mm.

Comme une tranche de CT est généralement reconstituée à 512 x 512 voxels, chaque tranche représente environ 370 mm de données en longueur et en largeur.

En utilisant les métadonnées de la DICOM, nous pouvons déterminer la taille de chaque voxel comme étant l'épaisseur de la tranche. Afin d'afficher le CT sous forme isométrique 3D (ce que nous ferons ci-dessous), et aussi de comparer entre différents scans, il serait utile de s'assurer que chaque tranche est échantillonnée en pixels et en tranches de 1x1x1 mm

```
id = 0
imgs_to_process = np.load(output_path+'fullimages_{}.npy'.format(id))
def resample(image, scan, new_spacing=[1,1,1]):
    # Determine current pixel spacing
    spacing = map(float, ([scan[0].SliceThickness] + list(scan[0].PixelSpacing)))
    spacing = np.array(list(spacing))

    resize_factor = spacing / new_spacing
    new_real_shape = image.shape * resize_factor
    new_shape = np.round(new_real_shape)
    real_resize_factor = new_shape / image.shape
    new_spacing = spacing / real_resize_factor

    image = scipy.ndimage.interpolation.zoom(image, real_resize_factor)

    return image, new_spacing

print ("Shape before resampling\t", imgs_to_process.shape)
imgs_after_resamp, spacing = resample(imgs_to_process, patient, [1,1,1])
print ("Shape after resampling\t", imgs_after_resamp.shape)
```

```
Shape before resampling  (236, 512, 512)
Shape after resampling   (118, 300, 300)
```

2-5- 3D Plotting

Il est utile de disposer de données isotropes, car elles nous donnent une idée de la dimension Z. Cela signifie que nous disposons maintenant de suffisamment d'informations pour tracer l'image DICOM dans l'espace 3D. Pour commencer, nous nous concentrerons sur le rendu des os.

Visualization Toolkit (VTK) est excellente pour la visualisation 3D car elle peut utiliser le GPU pour un rendu rapide. Cependant, nous n'arrivons pas à faire fonctionner le VTK dans Jupyter, nous allons donc adopter une approche légèrement différente :

- Créer une statique de haute qualité en utilisant la capacité 3D de matplotlib
- Créez un rendu de qualité inférieure mais interactif en utilisant plotly, qui est compatible avec WebGL via JavaScript.

L'algorithme **marching cubes** est utilisé pour générer un maillage 3D à partir de l'ensemble de données. Le modèle plotly utilise **step_size** plus élevé avec un seuil de voxel plus bas pour éviter de surcharger le navigateur web.

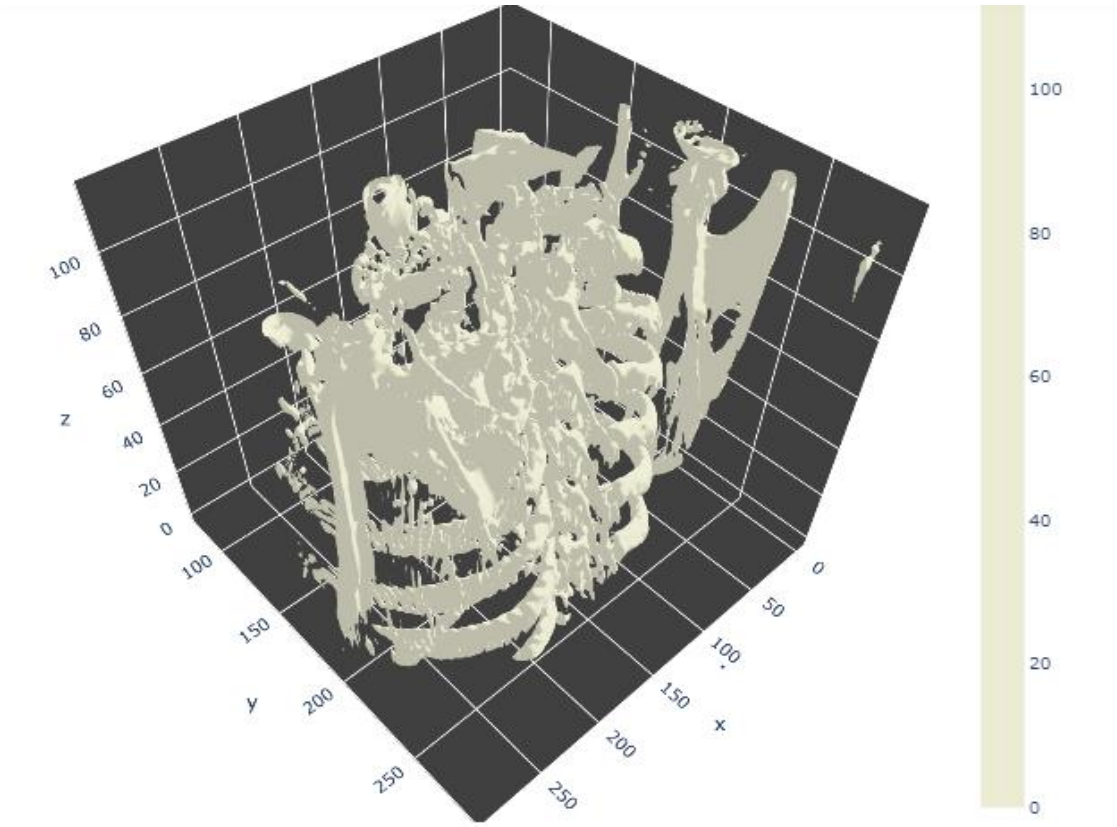
```
def make_mesh(image, threshold=-300, step_size=1):
    print ("Transposing surface")
    p = image.transpose(2,1,0)
    print ("Calculating surface")
    verts, faces, norm, val = measure.marching_cubes_lewiner(p, threshold, step_size=step_size, allow_degenerate=True)
    return verts, faces
```

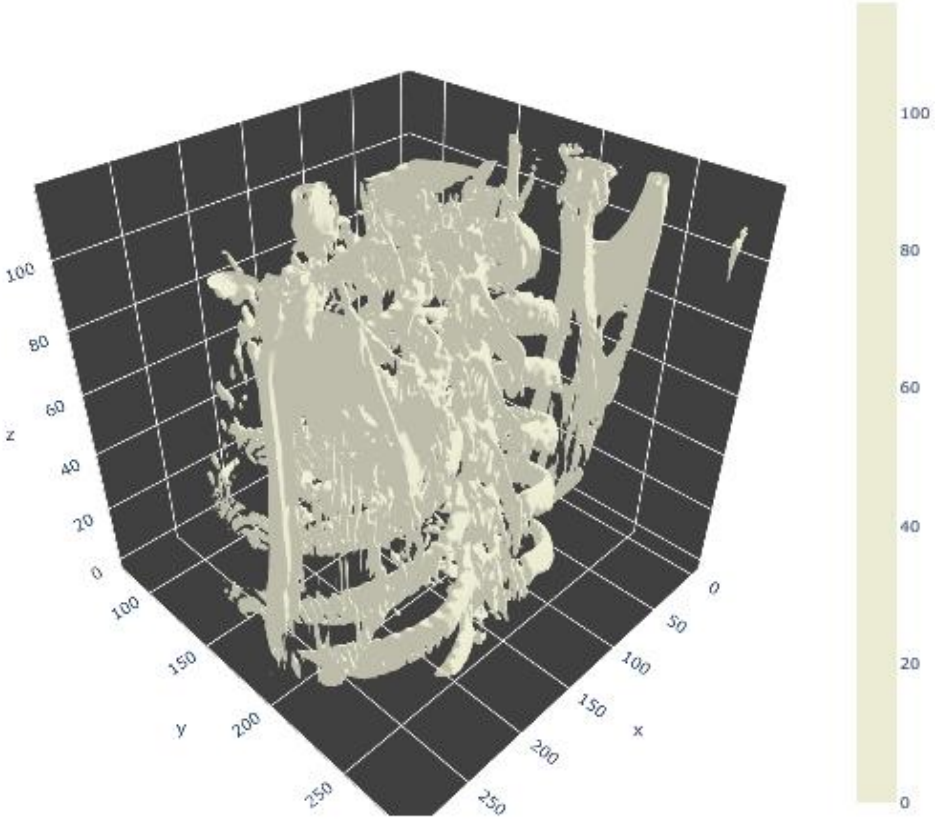
```
def plotly_3d(verts, faces):
    x,y,z = zip(*verts)
    print("Drawing")
    # Make the colormap single color since the axes are positional not intensity.
    colormap=['rgb(236, 236, 212)','rgb(236, 236, 212)']
    fig = FF.create_trisurf(x=x,
                            y=y,
                            z=z,
                            plot_edges=False,
                            colormap=colormap,
                            simplices=faces,
                            backgroundcolor='rgb(64, 64, 64)',
                            title="Interactive Visualization")
    iplot(fig)
```

```
def plt_3d(verts, faces):
    print ("Drawing")
    x,y,z = zip(*verts)
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection='3d')
    # Fancy indexing: `verts[faces]` to generate a collection of triangles
    mesh = Poly3DCollection(verts[faces], linewidths=0.05, alpha=1)
    face_color = [1, 1, 0.9]
    mesh.set_facecolor(face_color)
    ax.add_collection3d(mesh)
    ax.set_xlim(0, max(x))
    ax.set_ylim(0, max(y))
    ax.set_zlim(0, max(z))
    ax.set_facecolor((0.7, 0.7, 0.7))
    plt.show()
```

```
v, f = make_mesh(imgs_after_resamp, 350, 2)
plotly_3d(v, f)
```

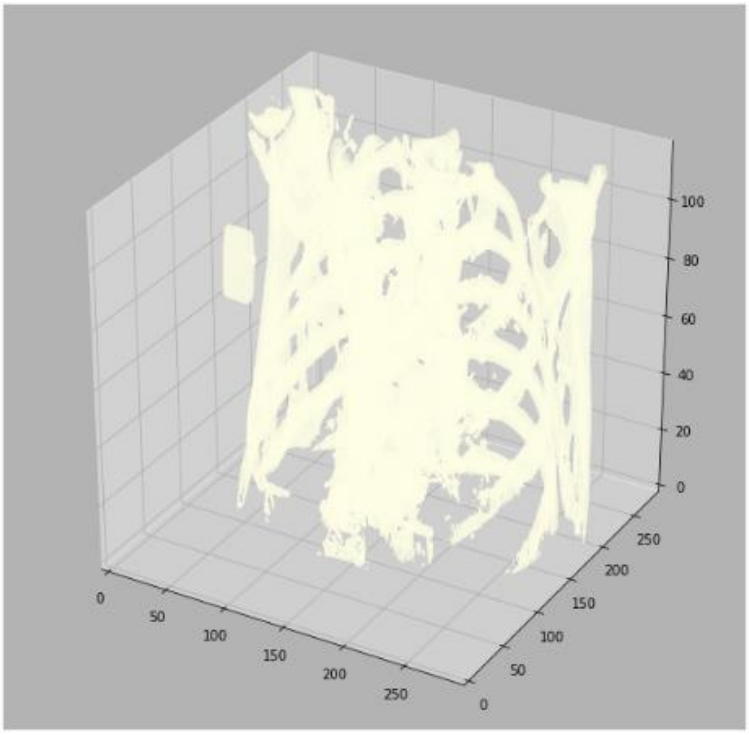
nous allons obtenir les résultats suivants :






```
v, f = make_mesh(imgs_after_resamp, 350)
plt_3d(v, f)
```

```
Transposing surface
Calculating surface
Drawing
```



3-6- Segmentation des poumons

Les algorithmes d'apprentissage automatique fonctionnent beaucoup mieux lorsque vous pouvez définir précisément ce qu'ils examinent. Une façon d'y parvenir est de créer différents modèles pour différentes parties d'un scanner du thorax. Par exemple, un réseau convolutif pour les poumons serait plus performant qu'un réseau général pour l'ensemble du thorax.

Par conséquent, il est souvent utile de prétraiter les données d'image en détectant automatiquement les limites entourant un volume d'intérêt.

Le code ci-dessous :

- Normaliser la valeur du pixel en soustrayant la moyenne et en divisant par l'écart type
- Identifier le seuil approprié en créant 2 groupes de KMeans comparant centrés sur les tissus mous/os vs les poumons/air.
- Utiliser **l'érosion** et la **dilatation**, qui a pour effet de supprimer de minuscules caractéristiques comme les vaisseaux pulmonaires ou le bruit.
- Identifier chaque région distincte sous forme d'étiquettes d'images séparées (pensez à la baguette magique dans Photoshop)
- Utiliser des cases délimitées pour chaque étiquette d'image afin d'identifier celles qui représentent les poumons et celles qui représentent "tout le reste".

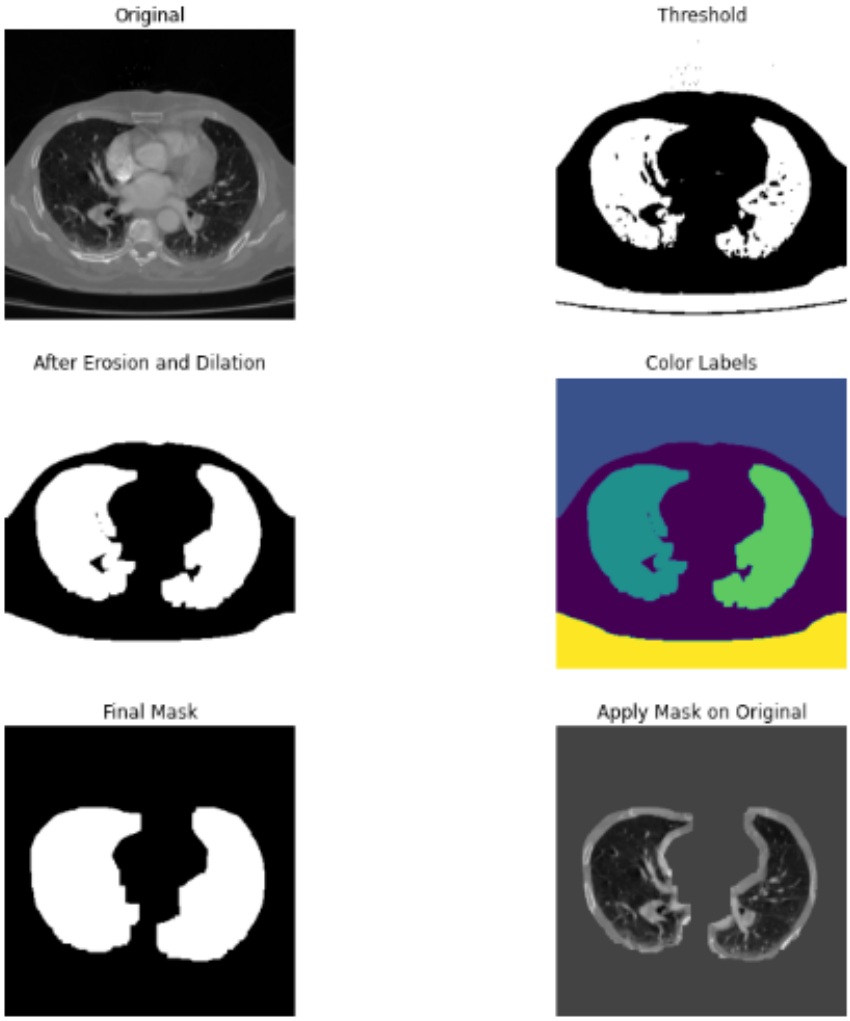
- Créer les masques pour les champs pulmonaires.
- Appliquez le masque sur l'image originale pour effacer les voxels en dehors des champs pulmonaires.

```
#Standardize the pixel values
def make_lungmask(img, display=False):
    row_size= img.shape[0]
    col_size = img.shape[1]
    mean = np.mean(img)
    std = np.std(img)
    img = img-mean
    img = img/std
    # Find the average pixel value near the lungs
    # to renormalize washed out images
    middle = img[int(col_size/5):int(col_size/5*4),int(row_size/5):int(row_size/5*4)]
    mean = np.mean(middle)
    max = np.max(img)
    min = np.min(img)
    # To improve threshold finding, I'm moving the
    # underflow and overflow on the pixel spectrum
    img[img==max]=mean
    img[img==min]=mean

    # Using Kmeans to separate foreground (soft tissue / bone) and background (lung/air)
    kmeans = KMeans(n_clusters=2).fit(np.reshape(middle,[np.prod(middle.shape),1]))
    centers = sorted(kmeans.cluster_centers_.flatten())
    threshold = np.mean(centers)
    thresh_img = np.where(img<threshold,1.0,0.0)  # threshold the image

    # First erode away the finer elements, then dilate to include some of the pixels surrounding the lung.
    # We don't want to accidentally clip the lung.

    eroded = morphology.erosion(thresh_img,np.ones([3,3]))
    dilation = morphology.dilation(eroded,np.ones([8,8]))
```

```
    labels = measure.label(dilation) # Different labels are displayed in different colors
    label_vals = np.unique(labels)
    regions = measure.regionprops(labels)
    good_labels = []
    for prop in regions:
        B = prop.bbox
        if B[2]-B[0]<row_size/10*9 and B[3]-B[1]<col_size/10*9 and B[0]>row_size/5 and B[2]<col_size/5*4:
            good_labels.append(prop.label)
    mask = np.ndarray([row_size,col_size],dtype=np.int8)
    mask[:] = 0

    #  After just the lungs are left, we do another large dilation
    #  in order to fill in and out the lung mask
    for N in good_labels:
        mask = mask + np.where(labels==N,1,0)
    mask = morphology.dilation(mask,np.ones([10,10])) # one last dilation

    if (display):
        fig, ax = plt.subplots(3, 2, figsize=[12, 12])
        ax[0, 0].set_title("Original")
        ax[0, 0].imshow(img, cmap='gray')
        ax[0, 0].axis('off')
        ax[0, 1].set_title("Threshold")
        ax[0, 1].imshow(thresh_img, cmap='gray')
        ax[0, 1].axis('off')
        ax[1, 0].set_title("After Erosion and Dilation")
        ax[1, 0].imshow(dilation, cmap='gray')
        ax[1, 0].axis('off')
        ax[1, 1].set_title("Color Labels")
        ax[1, 1].imshow(labels)
        ax[1, 1].axis('off')
        ax[2, 0].set_title("Final Mask")
        ax[2, 0].imshow(mask, cmap='gray')
        ax[2, 0].axis('off')
        ax[2, 1].set_title("Apply Mask on Original")
        ax[2, 1].imshow(mask*img, cmap='gray')
        ax[2, 1].axis('off')

        plt.show()
    return mask*img
```

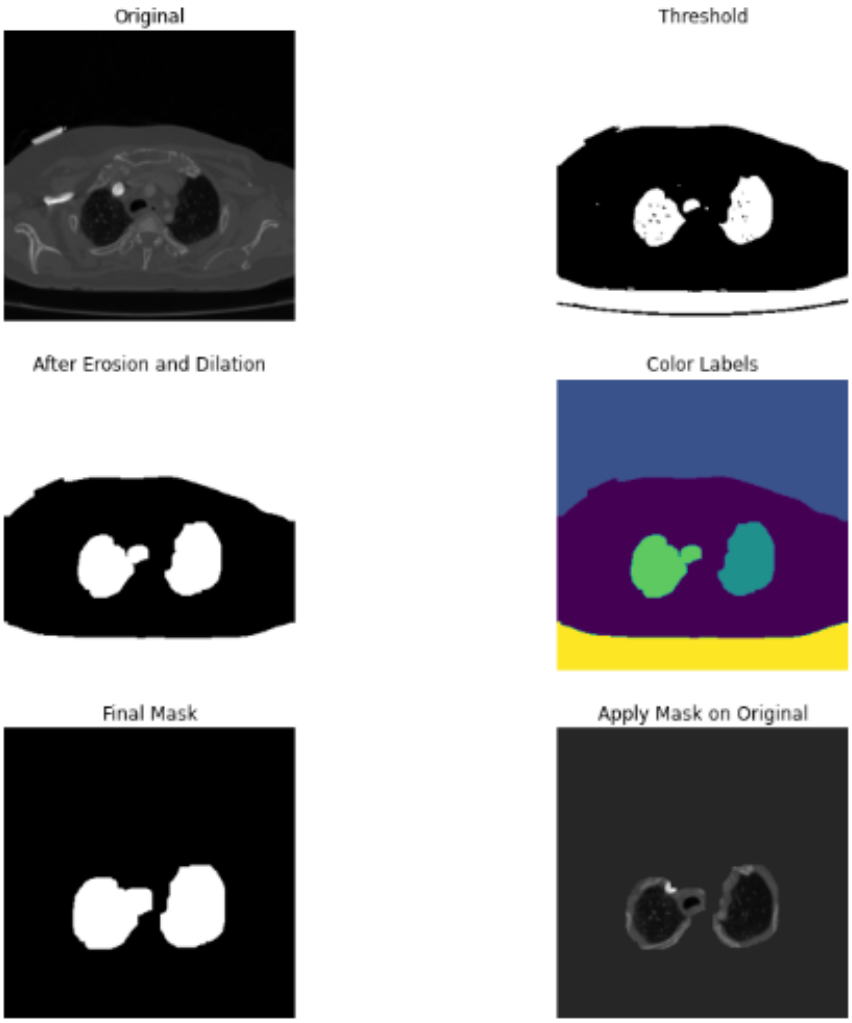
```
img = imgs_after_resamp[8]
make_lungmask(img, display=True)
```

Nous avons obtenu le résultat suivant du image 8 :




```
img = imgs_after_resamp[90]
make_lungmask(img, display=True)
```

Nous avons obtenu le résultat suivant du image 90 :



Quelques observations

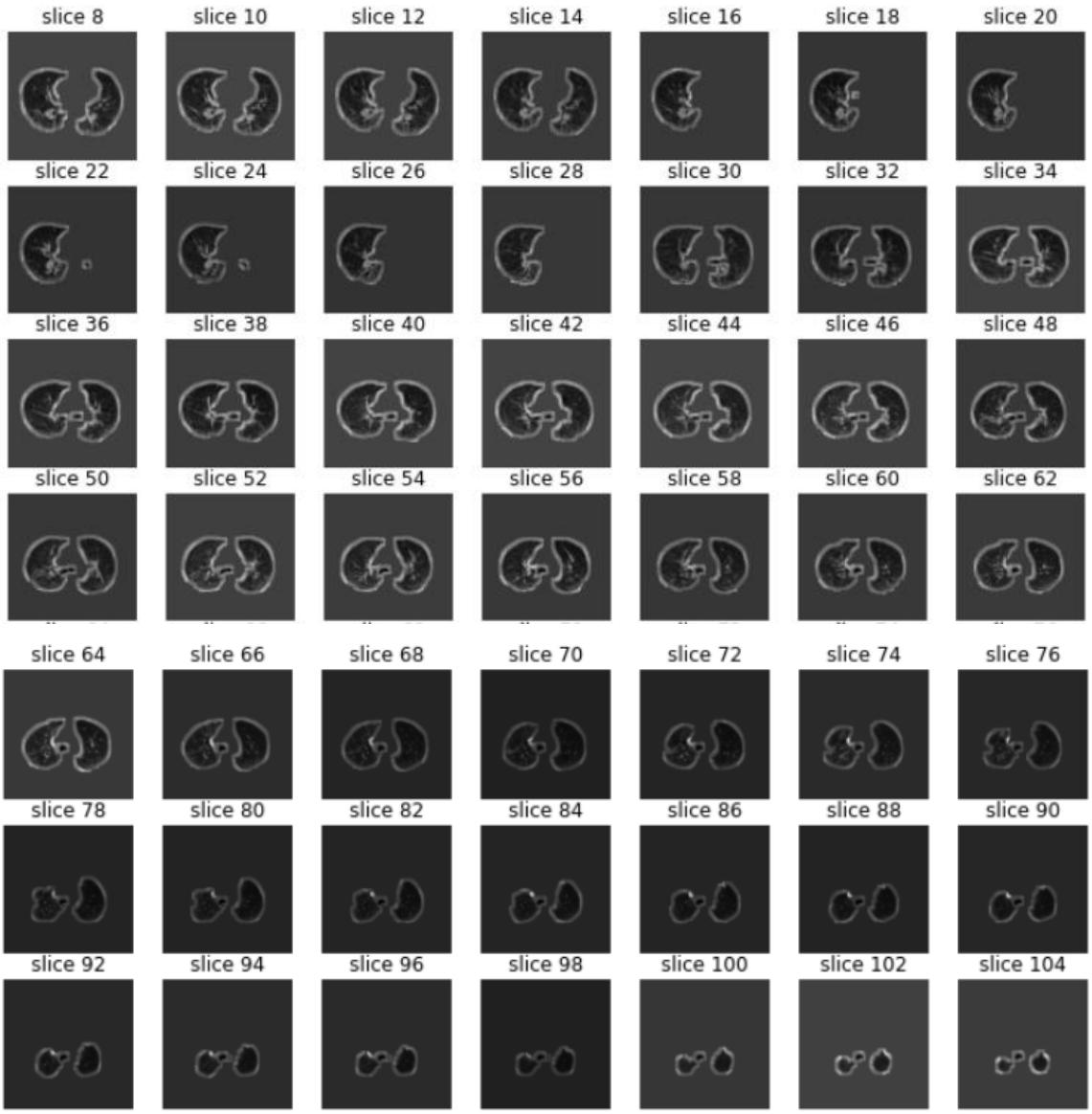
Si nous comparons la différence de contraste entre la tranche finie et l'original. Nous observons non seulement les données extrapulmonaires sont correctement nettoyées, mais le contraste est également amélioré.

Nous appliquons l'algorithme à toutes les tranches

L'exemple de la tranche unique semble fonctionner assez bien.

Nous appliquons maintenant le masque à toutes les tranches de ce scan .

```
masked_lung = []

for img in imgs_after_resamp:
    masked_lung.append(make_lungmask(img))

sample_stack(masked_lung, show_every=2)
```



Ce serait un bon moment pour sauvegarder les données traitées.

```
np.save(output_path + "maskedimages_%d.npy" % (id), imgs)
```

3- Segmentation et détection de l'embolie pulmonaire : L'algorithmme Mask_RCNN

3-1- Réseau de convolution basé sur les régions (R-CNN)

Le réseau de convolution basé sur les régions (R-CNN) est le premier travail qui a appliqué la méthode d'apprentissage en profondeur dans les problèmes de détection d'objet. L'idée principale est que l'algorithme trouve tous les objets dans une image en utilisant un algorithme de recherche exhaustive, puis classe les objets proposés en utilisant CNN. L'algorithme de recherche permettant de localiser des objets dans une image s'appelle recherche sélective. .[6]

Le modèle R-CNN combine la recherche sélective et les méthodes CNN pour localiser et classifier les objets. Le R-CNN est composé de trois modules. Le premier génère un ensemble de régions de proposition utilisant la recherche de régions d'intérêt. Le second est un CNN pour extraire un vecteur de caractéristiques de 4096 dimensions de longueur fixe de chaque région. Le troisième module est un ensemble de classificateurs SVM linéaires dont l'entrée est le vecteur de caractéristiques et sa sortie est la probabilité d'appartenir à une catégorie d'objet. L'architecture de R-CNN est montrée dans la fig 1.

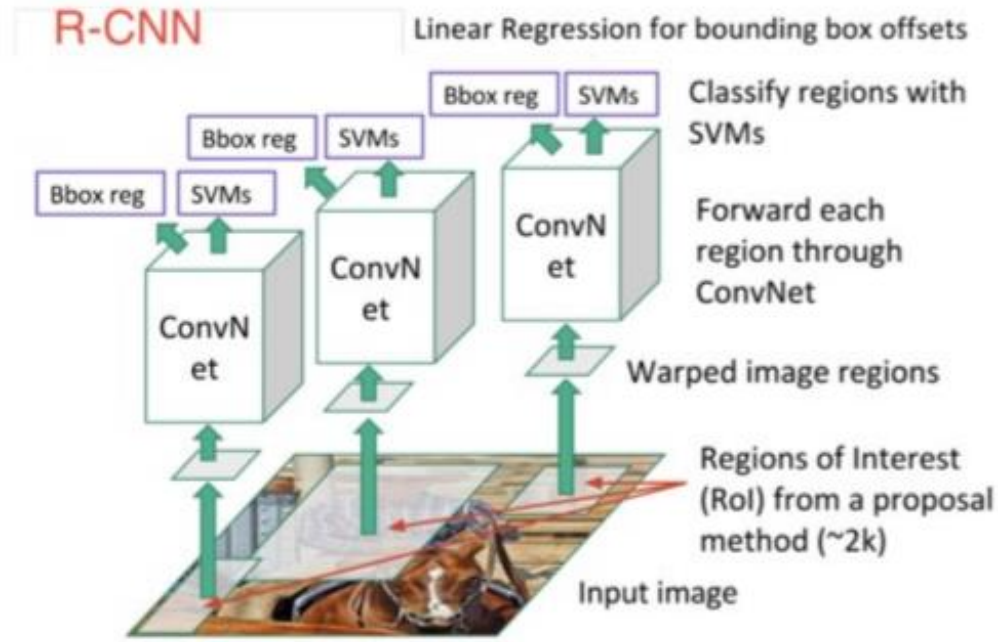


figure 2 :Principe du réseau de convolution basé sur les régions (R-CNN)

3-1-1 Fast R-CNN

Fast R-CNN est une variante de R-CNN visant à accélérer la détection d'objets.

Fast R-CNN est conçu pour réduire la quantité de calcul et de mémoire nécessaire à R-CNN en utilisant une perte multitâche pour entraîner l'ensemble du réseau en un seul passage et mettre à jour toutes les couches du réseau . L'architecture de R-CNN est montrée dans la

figure 2

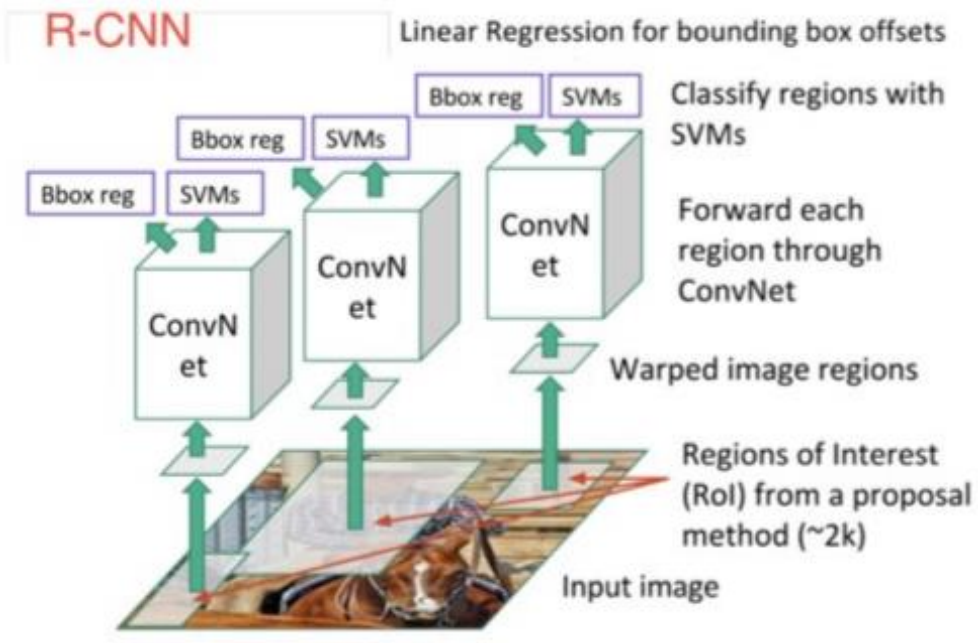


figure 3 :Principe du réseau de convolution basé sur les régions (R-CNN)

3-1-2 Faster R-CNN

Faster RCNN est une architecture de détection d'objets présentée par Ross Girshick, ShaoqingRen, Kaiming He et Jian Sun en 2015. Faster R-CNN est conçu pour remplacer l'algorithme de recherche sélective utilisé dans les versions précédentes de R-CNN. Le problème avec la recherche sélective est qu'elle est coûteuse en calcul. Bien que Fast R-CNN ait introduit de nouvelles fonctionnalités pour réduire le temps d'entraînement et de test, la recherche sélective restait un goulot d'étranglement pour les algorithmes R-CNN. Dans R-CNN plus rapide, un nouveau réseau appelé réseau de proposition de région (RPN) a été introduit pour remplacer l'algorithme de recherche sélective. Ce réseau vise à proposer des régions qui seront utilisées ultérieurement par le réseau Fast R-CNN pour prévoir les boîtes englobant et détecter les objets. RPN utilise un modèle préformé sur le jeu de données Image-Net pour la classification. Plus précisément, le réseau RPN, qui est un réseau de convolution profonde qui propose des régions, prend une image en entrée et génère en sortie une carte de caractéristiques. La carte de caractéristiques est ensuite utilisée par un petit réseau. . Il est à noter que Faster R-CNN fusionne le réseau RPN avec Fast R-CNN en utilisant un mécanisme appelé «attention mechanism». Le réseau RPN guide le réseau Fast R-CNN où chercher. Pour partager le calcul, les fonctions de convolution sont partagées entre RPN et Fast R-CNN. Le reste de l'algorithme est similaire à Fast R-CNN. Faster R-CNN est composé de 3 parties comme le montre la fig 3.[5]

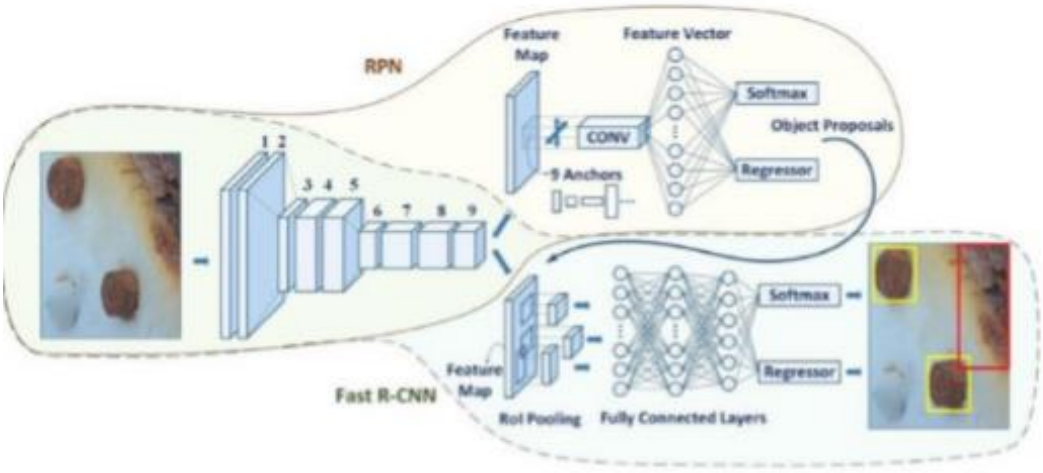


figure 4 :Architecture de Faster R-CNN

2- Réseau de convolution basé sur le masque de région (Mask R-CNN)

Le réseau basé sur le masque de région CNN (Mask Region-based Convolutional Network (Mask R-CNN)) étend Faster R-CNN en ajoutant une nouvelle partie à la reconnaissance du cadre de sélection afin de prédire un masque d'objet. Le masque R-CNN utilise les deux mêmes étapes que Faster R-CNN. Dans la première étape, Mask R-CNN adopte une architecture RPN identique, mais la deuxième étape vient avec une extension du Faster R-CNN d'origine. Dans la deuxième étape, outre les boîtes englobantes et les prédictions de classe, le masque RCNN produit un masque binaire pour chaque RoI.[5]

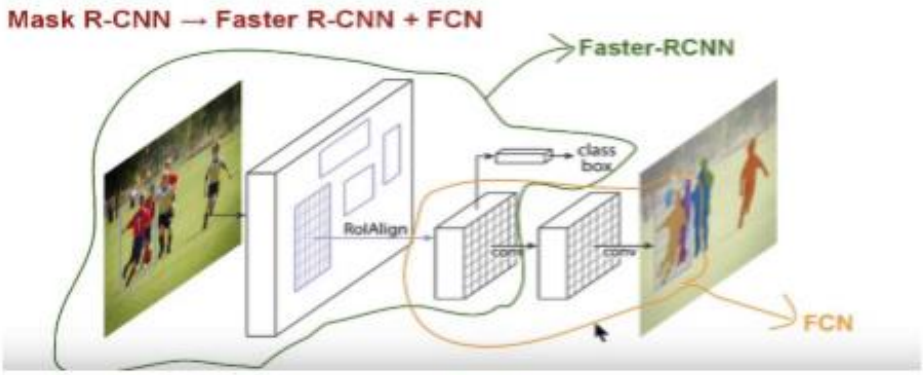


figure 5 :Architecture de Mask R-CNN

3- Application

Il s'agit d'une implémentation de Mask RCNN sur Python 3, Keras, et TensorFlow>=1.10, <2.0. Le code hérite de Mask RCNN. Nous avons principalement modifié sa partie RPN. Notre méthode est plus performante pour la détection de petits objets.

L'implémentation comprend :

- le code source de Mask RCNN
- le code source de la construction du modèle de mélange guassien (Guassian mixture model - GMM) et les implémentations de l'algorithme d'espérance-maximisation (Expectation-Maximization EM)
- le code source de l'échantillon provenant du Guassian mixture model(GMM)
- le code source du modèle RPN de P_mask_RCNN

Nous avons besoins des outils suivants :

numpy

scipy

Pillow

cython

matplotlib

scikit-image

tensorflow>=1.10.0,<2.0

keras>=2.0.8

opencv-python

h5py

imgaug

IPython[all]

C'est pour cela , nous avons installer le paquet suivant :

```
(base) C:\Users\Oumaima>cd Desktop\S2\traitement image\P_Mask_RCNN-master

(base) C:\Users\Oumaima\Desktop\S2\traitement image\P_Mask_RCNN-master>pip install -r outils.txt
```

sample.py est le module qui permet de construire des Guassian mixture model(GMM) et d'échantillonner des points à partir de GMM et de les enregistrer dans le répertoire locations .

```python
def sample_from_gaussian(mu, sigma, sample_No=1000):
    """
    Sampling from a Gaussian model
    :param mu: the mean of gaussian model
    :param sigma: the variance of gaussian model
    :param sample_No: the number of points that will sample from gaussian model
    :return: the sampled points
    """
    R = cholesky(sigma)
    s = np.dot(np.random.randn(sample_No, 2), R) + mu
    return s


def sample_from_uniform(alpha_vec):
    """
    sampling from a uniform
    :param alpha_vec: the probability each gaussian model being selected
    :return: the index of selected gaussian model
    """
    x = np.random.rand()
    total = 0
    index = -1
    for ids, a in enumerate(alpha_vec):
        if x >= total and (x < (total + a)):
            index = ids
            break
        else:
            total += a
    return index
```

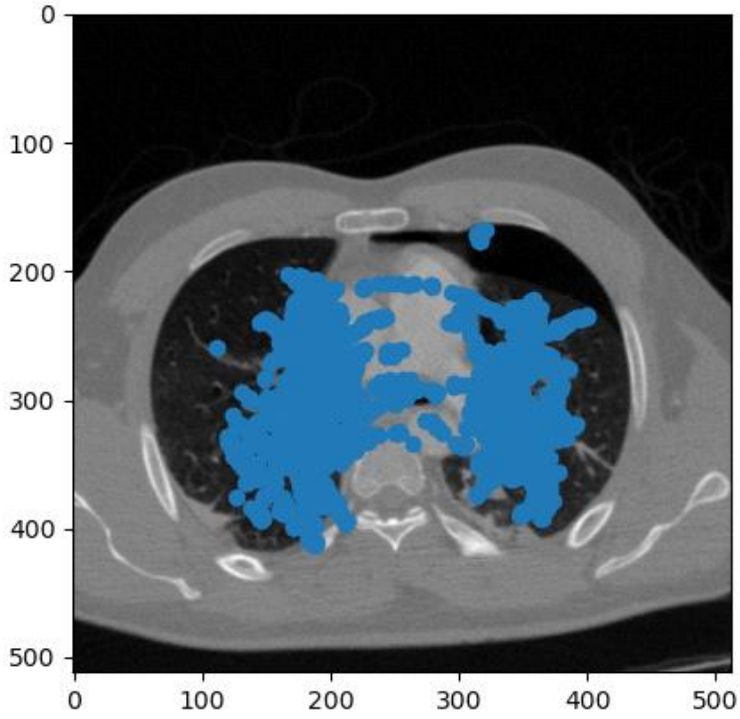
```python
def sample_from_mixture_gaussian(alpha, mu, sigma, sample_No=1000):
    """
    sampling from gaussian mixture model
    :param alpha: the weights of each gaussian model
    :param mu: the mean of each gaussian model
    :param sigma: the variance of each gaussian model
    :param sample_No: the number of points that will sample from gaussian mixture model
    :return: the sampled points
    """
    points = []
    for ids in range(sample_No):
        gaussian_ids = sample_from_uniform(alpha)
        p = sample_from_gaussian(mu[gaussian_ids], sigma[gaussian_ids], sample_No=1)
        points.append(p[0])
    points = np.round(points)
    # If the coordinate is repeatedly selected, then it is rejected.
    points = list(set([tuple(t) for t in points]))
    return np.array(points)
```

```python
def plot_object_distribution(points, image):
    """
    plot the location distribution of objects on the image
    :param points: objects' center coordinates
    :param image: background image
    :return:
    """
    plt.scatter(points[:, 0], points[:, 1])
    plt.imshow(image)
    plt.show()
if __name__ == "__main__":
    f = open("../annotations/instances_all.json")
    data = json.load(f)
    annotations = data["annotations"]
    centers = []
    for ann in annotations:
        bbox = ann["bbox"]
        y = bbox[0] + bbox[2] / 2
        x = bbox[1] + bbox[3] / 2
        centers.append([x, y])
    print(centers)
    centers = np.asarray(centers)
    image = plt.imread("../images/01123.jpg")
    plot_object_distribution(centers, image)
    gmm = gmm_em.GMM(centers)
    gmm.em_algorithm(100, 0.0001)
    print(gmm.mu)
    print(gmm.sigma)
    print(gmm.alpha)
    gmm_em.plot_gmm(gmm)
    alpha = np.array(gmm.alpha)
    mu = np.array(gmm.mu)
    sigma = np.array(gmm.sigma)
    points = sample_from_mixture_gaussian(alpha, mu, sigma, sample_No=100)
    np.savetxt("../locations/sample_points.txt", points, fmt="%d")
    print(points)
```

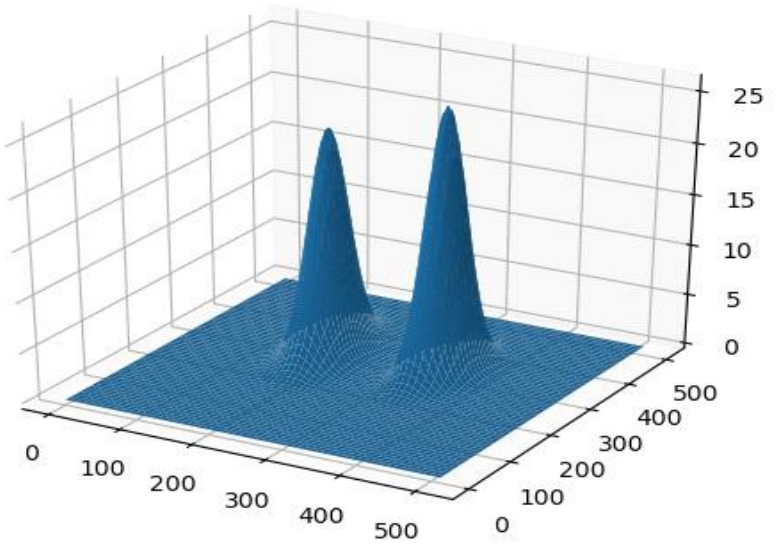
Nous allons exécuter le code et tracer le MGM dans un système 3D.

```
(base) C:\Users\Oumaima\Desktop\S2\traitement image\P_Mask_RCNN-master>cd Sampling

(base) C:\Users\Oumaima\Desktop\S2\traitement image\P_Mask_RCNN-master\sampling>python sample.py
```

les résultats sont les suivants :




```
err_mu: -0.00  err_sigma: 0.00  err_alpha: 0.00
[[184.6038  307.65283]
 [342.32983 312.35837]]
[matrix([[ 469.20557482, -261.80391687],
         [-261.80391687, 2066.32902361]]), matrix([[ 358.23605761,   36.33366422],
         [  36.33366422, 1467.17963146]])]
[0.5271547226602751, 0.4728452773397248]
2379.306302240195
1814.9728581876166
```



Pour former notre propre modèle, nous avons exécuté :

python pulmonary_embolism.py training --dataset="your dataset path" --model="your model weight file.h5"

```
(base) C:\Users\Oumaima\Desktop\S2\traitement image\P_Mask_RCNN-master>python pulmonary_embolism.py training --dataset="your dataset path" --model="your model weight fi
le.h5"
```

Pour évaluer le modèle, nous avons exécuté :

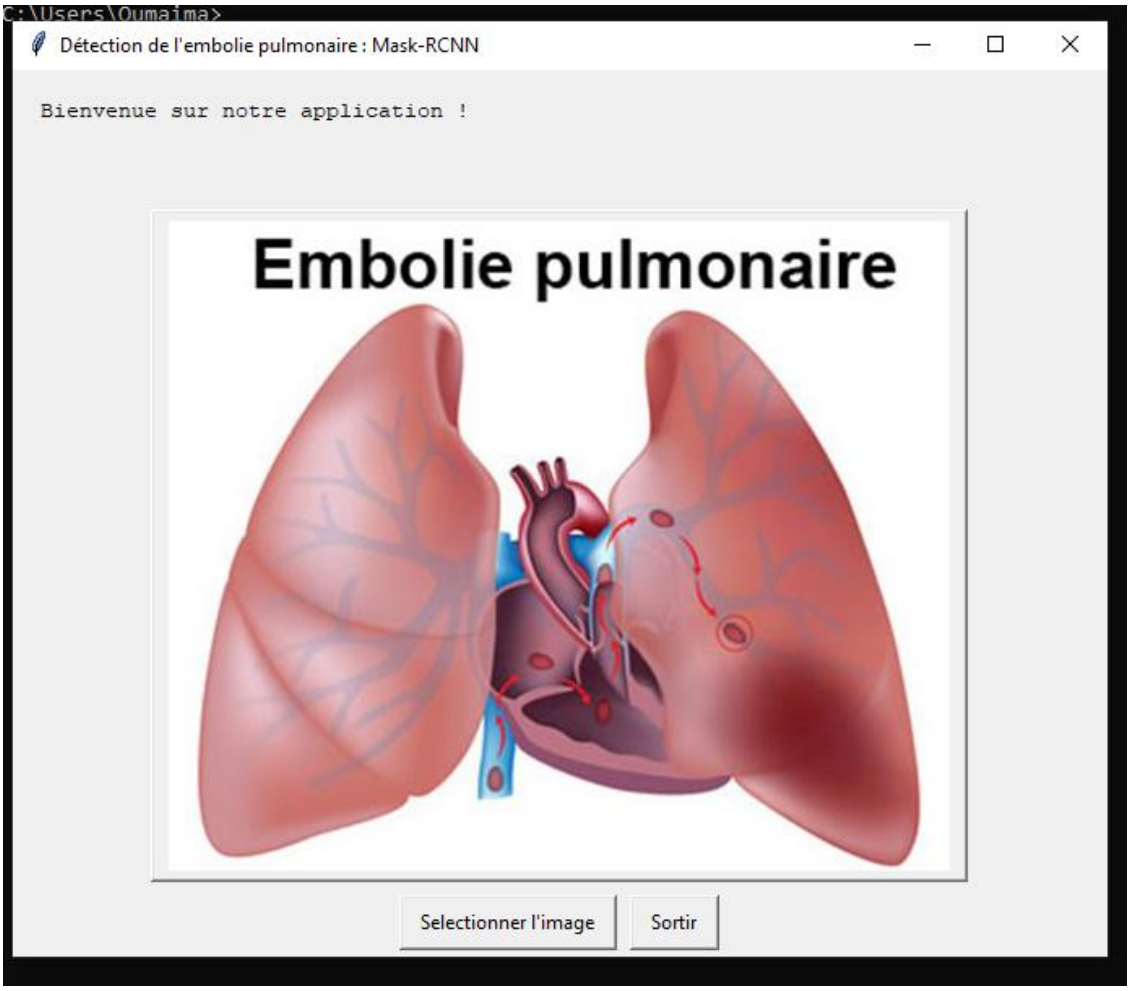
python pulmonary_embolism.py inference --dataset="your dataset path" --model="your model weight file.h5"

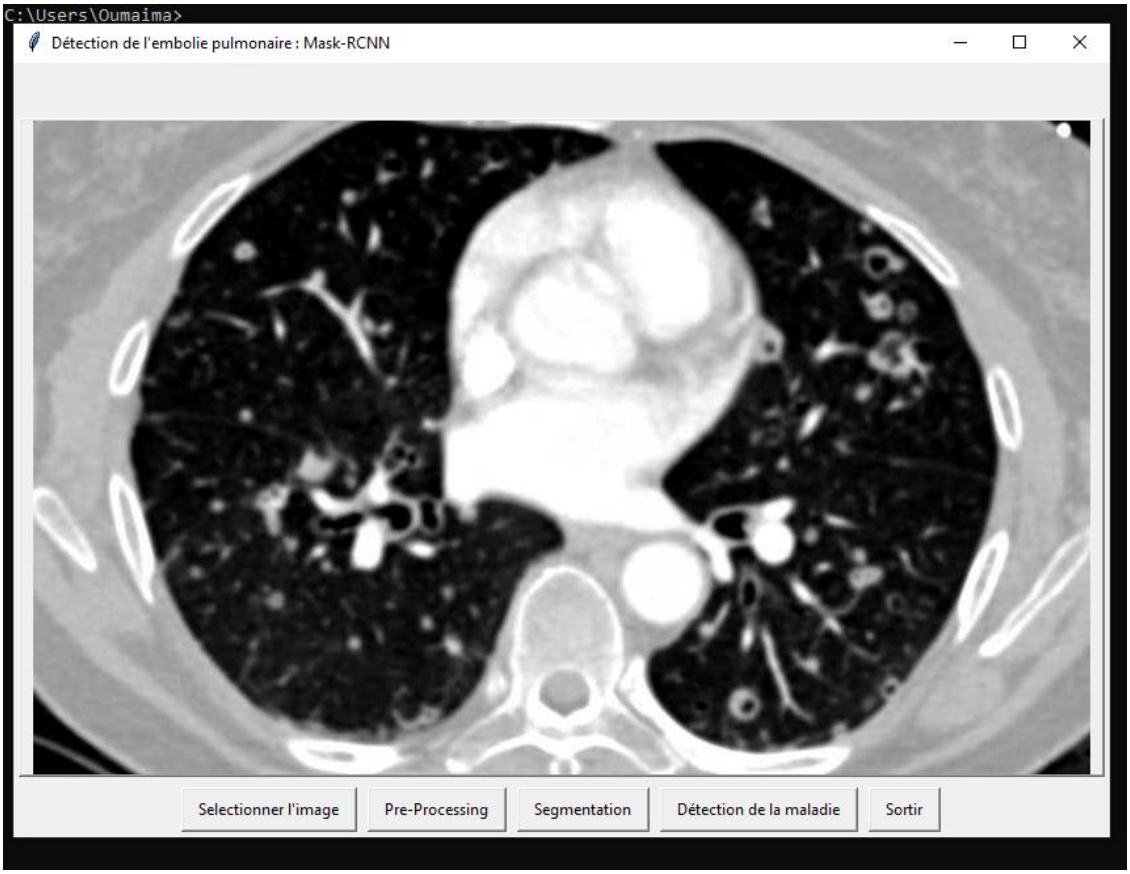
```
(base) C:\Users\Oumaima\Desktop\S2\traitement image\P_Mask_RCNN-master>python pulmonary_embolism.py inference --dataset="your dataset path" --model="your model weight f
ile.h5"
```

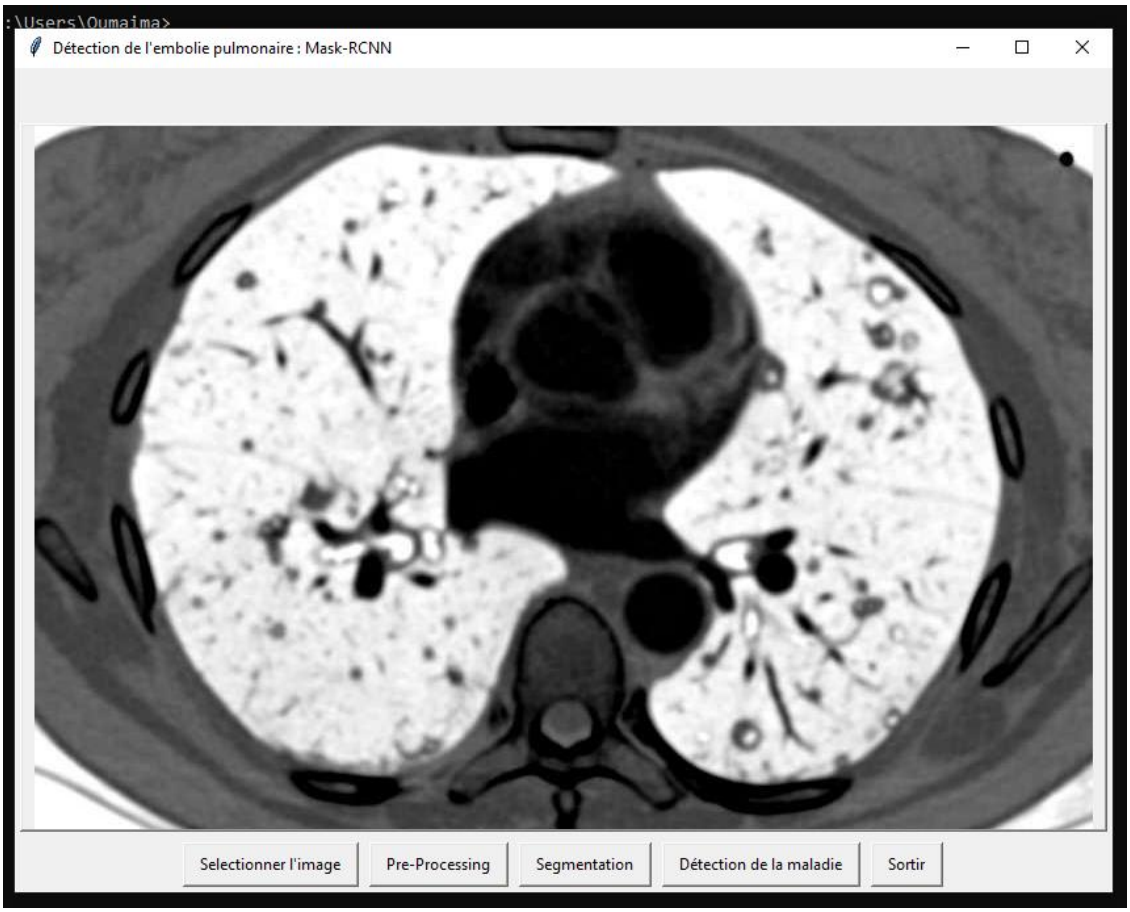
Perspectives

Nous voulons terminer l'application python que nous avons déja commencer, pour but de choisir l'image et d'appliquer l'ensembles des algorithmes du traitement pour obtenir l'image finale en détectant meme la maladie .

Les captures suivants montrent les étapes que nous avons réalisé:




C:\Users\Oumaima>
Détection de l'embolie pulmonaire : Mask-RCNN
Selectionner l'image
Pre-Processing
Segmentation
Détection de la maladie
Sortir


Détection de l'embolie pulmonaire : Mask-RCNN
Selectionner l'image
Pre-Processing
Segmentation
Détection de la maladie
Sortir

Réferences

[1] Thése du Doctorat :L'embolie pulmonaire massive Expérience du service de cardiologie de l'hôpital Moulay Ismail de Meknès :Etude rétrospective a propos de 10 cas- Mlle. Nadia LOUDIY -2008

[2] Apport de la tomodensitométriespiralée pour le diagnostic

d'embolie pulmonaire. Diehl JL, Perdrix L. Réanimation 2001 ; 10 : 71-5.

[3] Cours Embolie pulmonaire :Collège National des Enseigants de Réanimation Médicale

[4] Traitements interactifs d'images radiologiques et leursnapplications cliniques Julien Nauroy-2001

[5] Mémoire de Master :Localisation Indoor basé sur la détection Deep Learning.GHORMA Yahia . 2019